# Secure Invisible Computing

**Yong Xiong, Johannes Helander**
**Alessandro Forin, Gideon Yuval**

**7 October 2003**

**Technical Report**
**MSR-TR-2003-65**

Invisible computing creates better everyday devices by augmenting them with computation and communication. The native interface of a particular device suffices, with computation and communication occurring transparently to the user. An invisible computing device does not require setup or maintenance overhead and can be deployed incrementally without prerequisite infrastructure. Low-cost invisible computing devices could be used in areas such as home automation, wearable computing, sensor networks, or control of critical infrastructure, e.g. power grids.

Security is crucial in invisible computing. Nobody wants their home automation system accessed by others, or their everyday lives monitored by hackers. Security systems typically require the creation, distribution and revocation of security keys—a management chore that is potentially at odds with the invisibility requirements. Also at odds is the need for the devices to operate and communicate independently, without access to centralized services. Devices should also fail independently; compromising one device should not compromise the whole system. Finally, a personal security system should be able to scale and federate itself with other systems, for instance when some monetary transaction is involved.

This paper describes a security and communication model for invisible computing that combines limited resource consumption, interoperability, and security. It argues that the model presented sets a minimum level of functionality. It uses a combination of standard protocols and well known encryption primitives. The implementation for an embedded microcontroller demonstrates that the goals are achievable with an efficient and understandable design

**Microsoft Research**
**Microsoft Corporation**
**One Microsoft Way**
**Redmond, WA 98052**

# Secure Invisible Computing

**Yong Xiong*, Johannes Helander**
**Alessandro Forin, Gideon Yuval**

## ABSTRACT

Invisible computing creates better everyday devices by augmenting them with computation and communication. The native interface of a particular device suffices, with computation and communication occurring transparently to the user. An invisible computing device does not require setup or maintenance overhead and can be deployed incrementally without prerequisite infrastructure. Low-cost invisible computing devices could be used in areas such as home automation, wearable computing, sensor networks, or control of critical infrastructure, e.g. power grids.

Security is crucial in invisible computing. Nobody wants their home automation system accessed by others, or their everyday lives monitored by hackers. Security systems typically require the creation, distribution and revocation of security keys—a management chore that is potentially at odds with the invisibility requirements. Also at odds is the need for the devices to operate and communicate independently, without access to centralized services. Devices should also fail independently; compromising one device should not compromise the whole system. Finally, a personal security system should be able to scale and federate itself with other systems, for instance when some monetary transaction is involved.

This paper describes a security and communication model for invisible computing that combines limited resource consumption, interoperability, and security. It argues that the model presented sets a minimum level of functionality. It uses a combination of standard protocols and well known encryption primitives. The implementation for an embedded microcontroller demonstrates that the goals are achievable with an efficient and understandable design.

## 1 INTRODUCTION

Our everyday lives are filled with various devices that help us with our chores, provide entertainment, help improve our health, provide light and heat, and help us communicate. If we could improve those devices and make them work together—thus enabling new and better devices—our everyday lives would be more comfortable, social, and efficient.

Invisible Computing attempts to do just that. It combines everyday devices with computation and communication capabilities, enabling new functions and aggregation. It does this without forcing us to learn new or archaic computer interfaces. It makes it easy to adopt by keeping costs down and without requiring a pre-existing infrastructure. A watch could be used to control the volume on the radio. The refrigerator could automatically order more beer or notify the TV that its door is ajar. Energy could be saved by integrating sensors with the heating system. Wearable medical devices could improve our well-being. Utilities could reduce water lost from leaks. More natural user interfaces and ubiquitous communication would enable better social interaction regardless of our physical location. Smart toys would not only entertain, but also educate children.

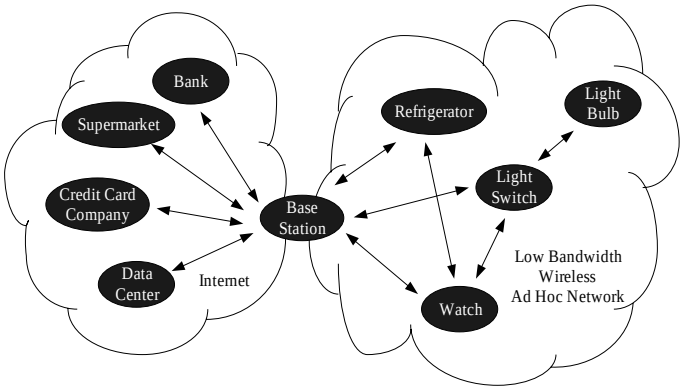



**Figure 1: The home as a secure, invisible computing system**

In virtually all of these applications we must pay attention to security issues. Installing a home automation system should not result in the loss of privacy. A wearable medical device could benefit from the ability to communicate directly with a doctor's office but should not leak sensitive biometric data to outsiders. Only the authorized physician should be able to tune a pacemaker. Supervisory control and data acquisition (SCADA) systems, which are used to control the operations of critical infrastructure (e.g. power utilities, distribution networks, and municipal water supplies), have generally been designed and installed with little attention paid to security [1].

An adequate security model for invisible computing devices must therefore ensure privacy and owner control at all times. To protect privacy we must use strong encryption for any and all communications. To protect owner control we cannot let the compromise of one device compromise the entire system. This requirement mandates a solid key exchange and trust management protocol; one that can only be implemented by a public key infrastructure (PKI), since access to authority servers is not always available.

Another issue that invisible computing systems face is interoperability. For instance, a home automation system must integrate cellular phones, refrigerators, TVs, heating systems,

* Currently Ph.D. Candidate at Texas A&M University

watches, heart rate monitors, etc. produced by different manufactures at different times. XML [30] Web Services [26] have been developed to help manufactures overcome this integration barrier. They provide loosely coupled, platform-independent, and language-independent application layer interoperability between different platforms, including small devices [3].

Combining the security and interoperability requirements with low cost and limited resources is a challenge. Current embedded systems either have no security or provide extremely limited interoperability.

This paper introduces a security model and an implementation that provides a solution to this challenge. It is designed for devices that often must operate in the absence of any infrastructure, communicating in an ad hoc fashion with a small number of peers over unsecured channels, such as wireless radio links or electrical power lines. At times these devices have access to general purpose computers or the Internet, and want to be part of the global service architecture.

The model provides service discovery, maintenance of trust/function relationships, server/client authentication, secure messaging, and local/global interoperability. It does not assume any global authority or availability of services, but still enables federating with external trust domains. Privacy and owner control are maintained via strong encryption and managed access at all times. Interoperability is achieved through a unified presentation layer that leverages XML Web Services. A base station is required when connecting to the Internet but not for peer-to-peer communication. Setup and discovery are integrated with trust establishment and key distribution—thus creating an easy to use, hassle-free user experience.

We implemented a prototype of this model on the MMLite platform [2][3]. The implementation uses standard protocols in an optimized fashion. The protocol uses a small number of the well known encryption primitives, RSA, AES, and SHA1; and a combination of the common protocols UDP, XML/SOAP [27][28][29], and Resurrecting Ducklings. In the Resurrecting Ducklings protocol, a device adopts the first "mother" it sees as its certificate authority. The implementation runs on a low-cost, single chip microcontroller [43] and performs well enough for real use.

Our main contributions are:

- We designed and implemented a security model for invisible computing systems that provides confidentiality, integrity, authentication, and interoperability. The model combines well-known protocols and algorithms in a novel and effective way, making it suitable for invisible computing.
- We use XML and SOAP as a general presentation layer and communication protocol. We use SOAP for trust management, key distribution, service discovery, two-way authentication, application communication, etc.
- We are proposing a trust relationship infrastructure that does not have hierarchy or a global or online *certificate authority* (CA). In our model, each person can have her own trust domain that is completely independent. Any two devices within a trust domain can securely communicate with each other without access to other devices or the Internet. Devices can set up occasional secure interoperation with the outside Internet and federate across domains, extending the security model to a global scale. A single sign-in is required instead of signing in to multiple domains. Our model combines well with the global WS-Federation specification [10] but differs from the Orange Book [22] in its lack of hierarchy.
- The model achieves both point-to-point and end-to-end secure sessions using the same security mechanism.
- Discovery, authentication, and key exchange are integrated together to save on memory footprint. Our simple service discovery protocol can work with and without other naming services. The discovery and bootstrap messages are encrypted just like any other message, thus privacy is not compromised at any phase of the system.
- Trust establishment and functional assignment are combined so that all setup chores are done in a single user interaction. The interaction is based on physical touch and predicts the correct action heuristically, drawing from any preceding interaction. For instance, first touching a light bulb when it is installed and then a light switch indicates to the light switch that it should control the light and it also gives it the authority to do so.
- The implementation shows that a reliable secure protocol based on PKI can be implemented on embedded systems, despite their limited resources. It offers further proof that using SOAP in embedded computing is a viable choice.

The rest of this paper is organized as follows: In sections 2, 5, 4, and 5, we present the security model and discuss various stages in achieving secure communication, including trust management, key distribution, authentication, and encrypted messaging. In section 6 we examine the cryptographic methods used in more detail. We discuss the implementation of the model in section 7, present a performance evaluation in section 8, review related work in section 9, and draw conclusions in section 10.

## 2 SYSTEM ARCHITECTURE

Our secure communication model logically comprises three stages. The first two enable the third.

1) **Trust relation establishment and role assignment**
   Who are the participants? This stage deals with questions on how a security domain is established; how an entity is created and what function it should have; how an entity is admitted into a security domain; and how to establish trust across domains.

2) **Discovery and key exchange**
   Why and how should we talk? Once an entity wants to talk to another entity it must first find its peer, verify trust, establish a communication key, and decide on communication routes and data representation.

3) **Service communication**
   What do we want to say? This stage is where the real work is done. Each entity presents itself as a service that can receive messages. The authenticity of the message needs to be verified and a decision made whether the message should be processed.

Each stage defines a set of protocols and policies. Each stage is explained in further detail in sections 3, 4, and 5. At all stages, basic security properties must be maintained:

- ***Confidentiality:*** *Ensuring data transmitted can not be read by unauthorized persons.* Messages are encrypted using AES and access to the keys is controlled.

- ***Authentication:*** *The process of verifying the identity of a data source and destination.* Certificates and public key cryptography of RSA and DSA are employed to show proof of a secret, and to establish peer-to-peer keys that are known only by the sender and receiver.

- ***Data Integrity:*** *Verification that the data received is accurate and un-changed by a man in the middle.* Message Authentication Codes (MAC) are used to provide message integrity. Cryptographic hash functions are calculated using SHA1. Sequence numbers prevent message duplication.

- ***Non-repudiation:*** *Proof of transmission and reception.* In most cases the knowledge of the peer-to-peer key is sufficient. If it is necessary to show that the receiver did not generate a message herself, digital signatures are calculated using PKCS#1 (SHA1 and RSA) or DSA.

- ***Access Control:*** *Permits or denies access based on attributes of server and client.* The certificate attributes are used for both mandatory access control (static decision) and for discretionary access control (dynamic decision).

## 3 TRUST MANAGEMENT

Trust management is central to the development of secure systems [40][22][23][4][18] and the application areas of our interest are no exception.

Trust is established between entities within a trust domain. The trust domain is controlled by an authority that is acknowledged by each member. Each entity is represented by a certificate and the knowledge of a secret that is only known by the entity itself. A certificate consists of the public key (RSA/DSA) that matches the secret and a number of attributes.

A small device—the focus of this paper—has one entity that represents the device. That entity is a member of one trust domain. Conversely, big computers may have multiple entities that represent various services or users on that computer, with each entity potentially being part of multiple trust domains.

In a consumer setting, each person can have her own *personal trust domain*. A personal trust domain consists of all devices that have direct trust relationship even if the device locates in the outside Internet. This means a personal trust domain is not limited to the local ad hoc network. The local ad hoc network can work independently from the outside Internet. Our model also supports occasional secure cross-domain interoperation. Cross-domain trust relations can be established through WS-Federation [10]. Unlike the orange book, our model is not hierarchical, as consumer use and a consumer society do not match a hierarchical model. Our model does not require a global *certificate authority* (CA), therefore systems can be set up completely independently and do not require connectivity.

### 3.1 Bootstrapping

Trust is established on a different channel than normal communication. In this paper, we call this channel *the secure channel*. This channel could be physical-touch based, meaning that any communication on this channel implies physical presence. It could be a personal visit to the bank, where a teller verifies that the driver's license matches the face. It could be a check with the hash of the public key. When the check clears, trust is established. The reception of money makes the bank trust the customer—the policy of what constitutes trust is up to the participants. The touch based scenario seems most appropriate for consumer electronic devices.

The trust is bootstrapped using the Resurrecting Duckling Protocol [5] on the secure channel. A device becomes a part of the family (the trust domain) of the first "mother duck" it sees. We say the "mother" is the trust authority of all devices in its

family. The "mother" is another device that signs the new device and the siblings believe in the signature because they share the "mother". We call the "mother" a *seal*, since it is analogous to the seal of a duke from old times. The seal can be viewed as a certificate authority (CA) of the domain. One seal defines a personal trust domain.

This is what happens: Initially a device is alone or "blank". Each device generates a key pair for itself. The seal creates a certificate for itself. A certificate is a public key and a collection of attributes that are signed by the seal's private key (SHA1 and RSA). The blank device sends its public key to the seal. The seal turns it into a certificate and sends it and its own certificate to the device. The device is now part of the family (trust domain). The attributes state what role the device should have, what the identity of the device is, and a contain a static access control list that—together with dynamic policies—provides fine grain control in later stages. Possible attributes include a name, a unique ID, a device category, access privileges, the initial location, associated devices, and the owner of the device.

The device proposes an attribute list, and the seal edits and signs the list together with the public key. The editing process is affected by what the seal touched previously and heuristically reflects the user's intent. For example, a user can touch a light bulb with her seal first and then physically touch a light switch. The light switch is associated to control the light and is authorized to do so. Not all switches need to be authorized to control all lights even if they all are certified by the same seal. Note that the physical touch is only needed when the light bulb is installed and later configuration can be done remotely if desired.

The seal also creates a shared *house key* and sends it to each of its siblings. This key is later used for discovery question messages to avoid any plain text communication.

The seal never needs to communicate over the public channel and does not need to be on-line after certificate initialization.

### 3.2 Reversion

A device with an established identity can also be reversed back to its initial ownerless state—the duckling is instructed by the seal to crawl back into its egg. This enables a new "mother" to gain control over a device after it has been sold. Before a device is reversed, the device will verify that the seal is its mother. The process is: The device generates a random number and encrypts it with its seal's public key. The seal then decrypts it with its private key. The device believes the seal is its mother only if the seal can recover the random number. After the mother is verified, the device simply purges its certificate and its seal's public key. It may be desirable that the reversion is signaled by flashing lights or similar action so that it cannot be done secretively.

In order to keep the private keys protected as much as possible, the private keys never leave the device. On the other hand, it is useful to be able to reverse a key even if the seal is lost—a backup would be desirable. Instead of giving out its own private key, the seal creates a certificate for a backup seal device with an ACL attribute that allows reversion. If the seal and the backup also sign each other's certificates, it would be possible to merge the trusts of each other by each step-sibling transitively verifying each other's certificates.

Certificate updates are handled similarly to reversions. Instead of simply purging the old certificate, the device generates a new certificate proposal and sends it back to its mother or backup mother for recertification.

Recall lists and other dynamic configuration changes can be sent over the public network at any time, with the limitation that changes will not immediately become active on disconnected devices. However, dynamic updates allow such functions as giving the cleaning lady temporary limited credentials to the home or canceling an old tenant with a single command.

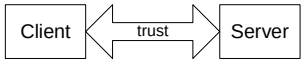
### 3.3 Direct Trust

Within a trust domain, all devices with the same owner trust each other directly. This is subject to the attributes in the certificate since they all trust their seal's signature. Direct trust can take place within the local ad hoc network or cross the Internet.

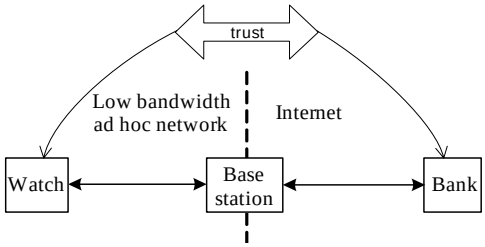
For the local case, shown in Figure 2 (a), each device has a certificate signed by the same seal and knows the seal's public key after bootstrapping with the Resurrecting Duckling protocol. Thus two devices can authenticate each other by exchanging their certificates.

Figure 2 (b) shows another case. Occasionally, one device wants to request a service outside of the local wireless network through the Internet. For example, a user wants to contact the bank from her watch. To set up a secure connection between the bank and her watch, a trust relationship must be established first. She could take her seal to the bank and use the touch-based authentication at the bank (the teller checks a picture of the customer and the customer sees the big building and believes it is a legitimate operation). The seal gives its own certificate to the bank and creates a certificate for the bank by signing the bank's public key and some attributes that state that this is the bank for the given account. The Resurrecting Duckling Protocol is used to achieve this. The process is essentially the same as when the seal admits a light switch.

The public keys could also be exchanged in other ways. In fact, rather than sending the entire key, a hash is sufficient. The customer could send a check with a hash of the public key. The bank could publish a hash of their key in a major newspaper. The attributes can be established manually based on the context. In a low-tech bank branch, the trust could be exchanged with ID cards and pieces of papers with the key hashes on them.




(a) Within ad hoc network




(b) Cross Internet

**Figure 2: Direct trust between devices within a personal trust domain**

The bank can later use the hash of the seal's public key to verify that a certificate sent to it is correct and signed by the seal. For example, a watch trying to contact the bank through a base station and the Internet will present its own certificate, send a copy of the seal's certificate, then request the bank's certificate (or use the manually entered certificate that came on a piece of paper).

### 3.4 Indirect Trust

In some cases, a device needs to request a service across trust domains. For example, the refrigerator wants to order some beer from a supermarket. The supermarket is not within the same trust domain as the refrigerator. There is no direct trust relationship between the refrigerator and the supermarket, but the refrigerator and the credit card company trust each other. The supermarket and the credit card company also trust each other. When the refrigerator wants to access the service of the supermarket, they talk to each other first to find which identity is trusted by both of them (step 1 in Figure 3). Then the refrigerator gets the public key of the supermarket from the credit card company (step 2 in Figure 3), as does the supermarket (step 3 in Figure 3), thus creating trust between the refrigerator and the supermarket.

The above case matches the Federated Identity Model of the industry-standard *global XML architecture* (GXA) [7][10]. In GXA terms, public keys are *security tokens*. A device acts as its own security token service instead of using a separate service. The level of trust established indirectly must be less or equal to either direct link.

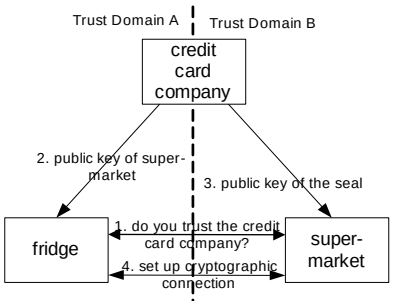



**Figure 3: Cross domain trust**

## 4 DISCOVERY AND KEY EXCHANGE

The purpose of this stage is to establish enough state to allow communication between two parties. The instigator of communication needs to find the proper entity to communicate with, authenticate the entity, establish a communication key, find a communication path to the entity, and negotiate the protocol and data representation.

For each peer an entity wants to communicate with, it caches a certificate with RSA and (optionally) DSA keys and attributes, a URL, a peer-to-peer AES and SHA1 key, a network route to where the peer was last seen, a sequence number, a session ID each way such as a UDP port number, and parameters (WSDL, compression, preferred protocol). Any of the state can be arbitrarily discarded with the cost of having to redo part or all of the discovery.

All communication in this stage is done on the public channel and is categorically encrypted.

### 4.1 Service Discovery

If the client wants to access a service, it will check if the URL of the service is cached. If not, it will send a *discovery request* through an IP multicast SOAP message on the public channel. The discovery request message gives a description about the requested service. This might be a DNS host name or something more abstract. The server sends back to the client a *discovery response* containing its URL through a unicast SOAP message. Potential service masquerading is detected by checking the certificate and ACL of the replying entity.

Sometimes a user needs to access a device from outside of her personal ad hoc network (e.g. the home). In this case, the user sends a service request to the base station of her personal network through unicast, and then the base station multicasts the service request within the wireless ad hoc network. The right service will send a *discovery response* to the base station and the base station will forward it to the client as a regular

internet directory service. The message forwarding is done with Web Service routing and the SOAP protocol. If the base station has a cache of the record of the required service, it could respond to the client directly. In this scenario, the base station acts as a discovery service.

Conversely, to find services that are outside the local ad hoc network from within the local network, the base station again acts as a proxy. The base station can either reply to the discovery message on behalf of the service, or the device can start by discovering a directory service and then use that for further queries.

All discovery messages (SOAP messages) are encrypted with a home key. The home key is an AES key generated by the seal and is common to the security domain. Its purpose is to keep curious neighbors from knowing what is being discovered. We will discuss AES use further in section 6.

### 4.2 Two-Way Authentication and Key Exchange

After a potential peer has been located, it is time for authentication and key exchange. Both parties authenticate each other. We use a combination of symmetric and asymmetric key encryption techniques that keep the asymmetric use at a minimum, since it is much costlier to calculate. The two-way authentication and peer-to-peer key exchange is done with RSA, while the connections are encrypted with an AES peer-to-peer key. If one of the parties has more resources, we then use DSA instead of RSA, as appropriate, to make the resource-rich party bear the brunt of the computation. This takes advantage of DSA's cost bias where public key operations are expensive but private key operations relatively cheap. RSA's cost properties are the opposite.

Authentication involves exchanging and verifying certificates. In a certificate, the public key of a device is the most important. It also contains the attributes of a device. Basically, authentication with PKI is a process to verify if a public key belongs to the right entity. If a public key is verified to belong to an entity, then one can trust that any information encrypted with the public key can only be understood by the entity with the right private key. The attributes are used to check against an Access Control List (ACL) to do authorization.

After the server and client authenticate each other, the client generates a random symmetric peer-to-peer key, encrypts it with its own private key and the server's public key (contained in the server's certificate), and sends it to the server. The server decrypts it with the client's public key (contained in the client's certificate) and its own private key. Generating a random peer-to-peer key and encrypting it with the private key can be done during idle time prior to the key exchange. This avoids latency required for encryption with a private key. A large number of random keys can be pre-computed this way when convenient. Caching the results turn these public/private key operations into one-time events.

## 5 SERVICE COMMUNICATION

Once a shared peer-to-peer key is created, an encrypted connection can be set up between the client and the server. Encryption is applied to SOAP messages instead of transport layer packets like SSL does. Every node is a server and every node accepts SOAP messages.

There are four basic cases of communication patterns:

1. Direct communication between peers in an ad hoc network.

2. Communication to the outside where the base station is trusted with the data.

3. Communication with the outside where the base station is not trusted.

4. Communication within the ad hoc network through an intermediary.

In all cases, communication comprises encrypted SOAP messages. How the messages are encoded and what parts of the messages are encrypted with what keys vary slightly.

A SOAP message consists of a SOAP header and SOAP body. The method name and parameter values of a remote call are contained in the SOAP body. The SOAP header has information relevant to the messaging itself, in particular the URL of the service object the message is meant for and a list of any intermediaries that are needed in delivering the message. The SOAP body should only be readable by the final server and the client, yet unreadable to any intermediate nodes. The information meant for the intermediaries should only be legible to the correct entities, thus the SOAP header needs to be encrypted as well.

### 5.1 Peer-to-Peer Messages

The discovery process established a peer-to-peer key and a session identifier (SID). The sender calculates a digital signature using SHA1 HMAC and attaches it to the message. The sender then encrypts the entire message (excluding the SHA1 HMAC value) using AES in the CTR (counter) mode and attaches the SID and sequence number in plain text to the beginning of the message.

The AES encryption is driven by the sequence number and the peer-to-peer key. The CTR mode requires a unique bit pattern for each block. The sequence number is incremented for each block and used as the index. The instigator of the discovery

uses odd numbers and the respondent uses even numbers, avoiding the generation of the same sequence number. Once the bits in the sequence number have been exhausted or for any other reason either of the peers is free to discard the peer-to-peer key and force a new key exchange to take place.

The receiver gets the message and uses the SID to look up the correct peer state (if none is found the message is discarded). The received sequence number is compared with the expected sequence number and if it is smaller, the message is assumed to be a duplicate and is discarded. The expected sequence number is, however, only updated once the message has been accepted so as to prevent simple denial of service attacks.

Next, the receiver applies the sequence number and peer key to decrypt the message. It then calculates a HMAC checksum using the peer key and compares it to the one received. If the checksum does not match, the message is discarded.

Finally the receiver checks the message against the attributes of the peer and decides whether the message should be processed in light of access control lists and possible recall lists, etc. If the message can pass through all the filters, it is processed by the SOAP deserializer and served by the correct server object.

The reply is sent back to the client with information in the SOAP routing header that lets the client correlate the response with the request. All the encryption and verification is done exactly like the request, but sent in the opposite direction.

Note that it is safe to transmit the SID and sequence number in plaintext. Any forgery inevitably leads to the message being discarded by the receiver as the AES and HMAC calculations will not yield the expected result. The information content of these numbers is low enough that it does not constitute a privacy threat. The SID could be a UDP or TCP port number thus saving a couple of bytes.

### 5.2 Messages through Trusted Base Station

Communication with the outside world entails interoperability and a base station. The simplest case is where the base station is trusted enough to handle low security operations and can offload complications from a device. An example scenario is a refrigerator that wants to order milk from the supermarket. The refrigerator is no more secure than the base station so the base station can order the milk on behalf of the refrigerator.

The refrigerator authenticates itself to the base station and uses the peer-to-peer communication patterns. The base station then uses the WS-Security and WS-Federation Internet Web Service protocols encoded as XML and Base64 blocks to order milk from the supermarket, while the wireless side simply encrypts the entire message. The two communication patterns and trust relationships are created independently. This scenario precisely corresponds to the *Passive Profile* of WS-Federation.

### 5.3 Messages through Untrusted Intermediary

In some cases, the security of the base station is insufficient; therefore an end-to-end secure channel must be created. The base station is still needed for forwarding messages. This corresponds to the *Active Profile* of WS-Federation.

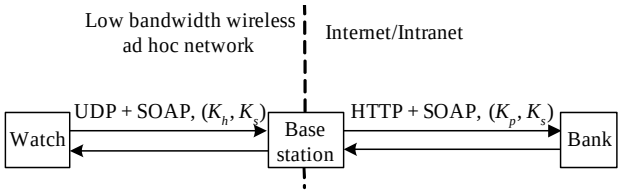



**Figure 4: A watch sends messages to a bank**

The scenario in Figure 4 is an example of this. The watch sends a SOAP message to the bank through a base station. The watch is within the wireless network and the bank is on the outside Internet. The base station knows how to forward the message to the bank. The base station needs to access the SOAP headers but only the bank and the watch need access to the body.

The SOAP body is encrypted with the end-to-end (watch to bank) peer key ($K_s$ in Figure 4). Only the endpoints can understand the contents of the body. On the wireless link the SOAP header is encrypted with the peer key that is shared between the base station and the watch ($K_h$ in Figure 4); while between the base station and the bank, the SOAP header is encrypted with a peer key ($K_p$ in Figure 4) that is shared between the base station and the bank. Since the base station and the bank have more powerful processors than the watch, they can bear the burden of the key exchange (see section 6.1).

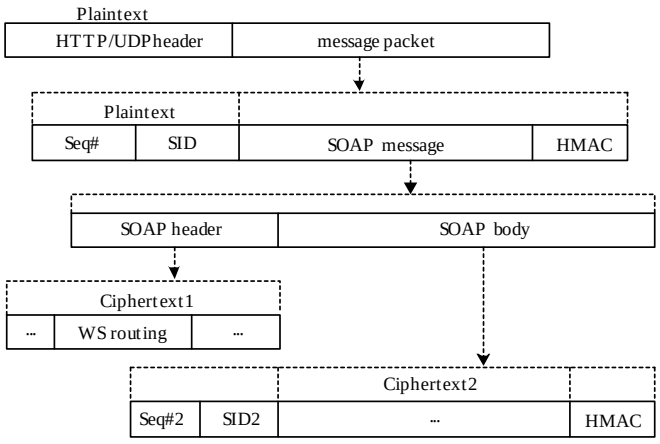



**Figure 5: A routed message packet**

SOAP messages can theoretically be sent over any transport. Our implementation supports TCP/HTTP and UDP. Within the wireless network, SOAP messages are transmitted through UDP instead of HTTP. UDP is much cheaper than HTTP and the extra headers are not needed. The base station then forwards the SOAP message to the bank through HTTP. The message forwarding is done using a Web Service routing SOAP header.

Figure 5 shows the format of a message packet. Since it pointless to encrypt the same data twice, the "inner" packet is only encrypted with the end-to-end key; while the "outer" packet is encrypted with the point-to-point key. The outer HMAC, however, covers the entire message to maintain integrity between the halves.

### 5.4 Ad Hoc through Intermediary

This case is essentially the same as communication with the outside. It may involve the base station as a more powerful and possibly trusted intermediary or it may involve any device that happens to be able to do forwarding. Security can be constructed either end-to-end or hop-to-hop as needed.

Scenarios could be arbitrarily complex where multiple headers should be understood by multiple intermediaries. The encoding model presented here extends to more complex cases and the small number of primitives presented here can be expected to cover all practical uses.

## 6 CRYPTOGRAPHIC PRIMITIVES

In this section, we examine the particular ways in which the cryptographic primitives RSA, AES, SHA1, and optionally DSA are used in our model. We use two specific optimizations to improve performance. First, we *pre-calculate encryption functions* allowing pipelining that reduces latency. Second, we recognize that sometimes there is a *strength imbalance* between two peers. In that case, we let the stronger peer carry a heavier calculation burden.

### 6.1 Public Key Primitives

The protocols in this paper use public key cryptography in four places:

1. Signing of certificate by the seal.

2. Signing peer-to-peer key to prove it came from the correct entity.

3. Encrypting peer-to-peer key in such a way that nobody but the correct receiver can eavesdrop the message. The receiver proves by understanding the message that it is the correct entity.

4. Signing of message for which it is important to prove that the receiver did not generate the message herself. This might be instructions for a monetary transaction.

In the cases 1, 2, and 4, we use a digital signature algorithm. The sender uses her private key to sign the message. The sender uses the sender's public key to verify it. PKCS#1 [13] is used usually, with SHA1 for hashing and RSA for signing. In case 3, the key can be pre-generated and signed. When the key is needed, the pre-generated key is used—saving the latency of the RSA private key operation. The pipelining process reduces encryption latency significantly (see Table 2), since RSA is biased in such a way that public key operations are much faster than private key operations.

In case 3, the receiver must do the private key operation on an unknown secret and thus cannot pre-calculate using RSA. If one of the peers is more powerful or has more available energy than the other, the key generation should be instigated by the weaker peer so that the expensive calculation can be done by the stronger peer. If the strong peer is the instigator, the receiving weak node discards the proposed key and instead generates its own key and sends it back to the instigator.

In case 4 when the sender is a strong node, we again use PKCS#1. If the sender instead is a weak node, we use DSA (ElGamal), where the private key operation can be pre-calculated without the message [41]. The DSA public key operation that is more expensive is done on the stronger node. It would be possible for the seal to use DSA for signing the certificate thus reducing latency. Unfortunately, that latency would then be added to every certificate validation—not necessarily the best tradeoff (once per device vs. once per device discovery).

The two remaining expensive operations are not extremely frequent: The certificate signing is done once per device. The peer-to-peer key exchange is done once per device pair when they communicate for the first time.

### 6.2 Symmetric Encryption

The CTR mode algorithm is shown in Figure 6. We use a 128-bit CTR mode [15] [16] AES cipher to encrypt messages. The CTR mode provides chaining of blocks, which is more secure than encrypting each block individually.

There are four other reasons why we choose the CTR mode. First, the CTR mode algorithm provides semantic security. Second, in CTR mode, only the sequence number of a message, instead of individual counters for blocks, needs to be transmitted. All counters can be easily constructed from a message's sequence number. Third, a sequence number is needed in the message anyway to detect duplicate or delayed messages and the same number can be used for AES. Fourth,

the counters can be predictable and encrypting/decrypting a message is independent from other messages. The CTR mode is defeated only if a counter is repeated. There is no lower limit to the length of the counter, but if the bits run out, a new peer-to-peer key must be generated.

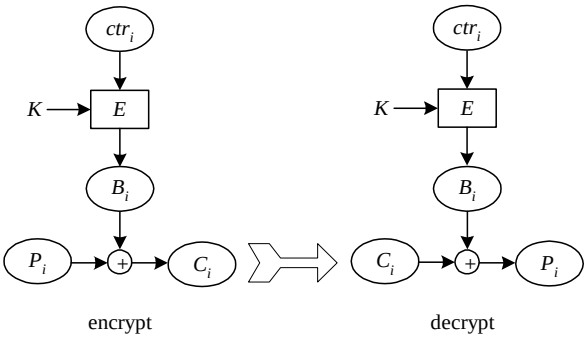



**Figure 6: Counter mode block cipher. E stands for any symmetric cipher function, e.g. AES, K a secret key, $ctr_i$ the $i^{th}$ counter. $P_i$ the $i^{th}$ block of plaintext, $C_i$ the $i^{th}$ block of cipher-text, and $B_i$ the result of applying E on $ctr_i$. $\oplus$ stands for the XOR operator.**

Since the counter is predictable, the CTR mode lends itself well to pipelining. New encryption pads are calculated during idle time after previous messages have been processed. At latency critical times, when a server needs to respond to a request message, only an exclusive-or (XOR) operation is needed. This way AES does not factor into latency, but only to the frequency that messages can be processed. Also, the pipelining does not save energy but the pre-calculation could be done more aggressively when more energy is available if there is fluctuation.

### 6.3 Hashing Primitives

Encryption itself is not enough to keep the integrity of the data. Integrity is verified by a MAC (*Message Authentication Code*). This is a bit string that is a one way hash function of the message and the secret key. The secret key is shared by the sender and the receiver and is part of the peer-to-peer key.

The SHA1-checksum algorithm is used with the symmetric key MAC [14] when the sender and receiver trust each other. When it is important to prove which peer generated the message, a more expensive digital signature that uses SHA1/RSA or DSA is used instead.

## 7 IMPLEMENTATION

We implemented a prototype of the model described in this paper on the MMLite platform [2][3]. The platform is specifically designed for invisible and embedded use. It takes an objects everywhere approach and is built out of components that expose well-defined interfaces. The platform combines the benefits of special and general purpose systems. It is general-purpose in the abstract, allowing code reuse and quick programming, but special purpose in the concrete, allowing efficiency and small footprint. The same interfaces are implemented by many components. These components are rarely aware of their intended system layer, and the same implementation can be reused in many different places.

MMLite adopts a unified namespace to hold any kind of object, e.g. a file, a heap, an instance of any class or even a C struct. Through this unified namespace, a component can gain access to objects maintained by other components. The namespace has a close relationship with Web URLs. All components and interfaces are described by an XML metadata database that is also used to drive SOAP serialization and deserialization.

MMLite supports programming in a convenient C environment for performance critical components or in C# when desired for code mobility or quick extensibility.

MMLite uses SOAP as a general communication protocol and XML for data representation. For example, we use SOAP to do key distribution, service discovery, two-way authentication, etc. The implementation interoperates with ASP+ and the SOAP Toolkit on Windows XP®. The SOAP implementation supports high level remote method calling. Through automatically generated SOAP proxies it is almost as easy to call a remote method as it is making a function call in a local application, with some obvious differences in timing and failure modes.

The network component is based on the Berkeley 4.4BSD-Lite TCP/IP stack. The stack is adapted to share network buffers with other components. This way the same buffer is operated on in different parts of the system without copying, and saves significant amounts of memory. The driver puts the data into a buffer, which is operated on by the network stack, is directly decrypted, and the XML parser and SOAP deserializer use the same data.

Besides the core system components, any networking, security, and application code is encapsulated into components. In an embedded system there is no real distinction between an application and an OS kernel, and this is reflected in the system. Components are insulated only when needed by security reasons. Insulation for security is not confused with the modularity of the system.

We use a pipelining technique to save latency. This is done in peer-to-peer key generation when encrypting with a private key (see sections 4.2, 6.1). Latency is also saved in prediction and encryption of counters in CTR mode AES cipher (see section 6.2).

## 8 PERFORMANCE

We measured our system on AT91EB63 evaluation boards (called EB63 board for simplicity in the rest of this paper) [42]. An EB63 has a 25MHz ARM7 microcontroller, 256Kbyes SRAM and 2 Mbytes Flash. Since this is more than a cost-effective system would have, we limited the memory usage to 32KB of RAM and 256KB of Flash ROM. The board was chosen for its processor and I/O capabilities, not for its energy efficiency or lack thereof.

Instead of an actual wireless network we ran the measurements over serial lines with one serial line representing a secure channel and another representing the public channel. The serial lines were run at 38400 baud, which is in range of several available low-power wireless radios. We used a PC as a "base station" achieving connectivity to the Internet.

We will now evaluate the cost of the different pieces based on measurements. The cost includes costly system resources, time, and energy.

| Files | ROM | Static RAM | Heap | Stack | Total RAM |
|---|---|---|---|---|---|
| BASE | 24,676 | 1,940 | 2,837 | | 2,777 |
| DRIVERS | 11,464 | 332 | 896 | 2,288 | 3,516 |
| NET | 77,024 | 3,424 | 2,648 | 3,400 | 9,472 |
| XML | 7,860 | 16 | 88 | | 104 |
| SOAP | 29,504 | 280 | 996 | 4,320 | 5,596 |
| SECProto | 14,180 | 604 | 1,848 | 2,648 | 5,100 |
| AES | 16,532 | 8 | | | 8 |
| RSA | 9,784 | 28 | 24 | | 52 |
| SHA1 | 5,436 | 8 | | | 8 |
| C-Library | 7,620 | 12 | | | 12 |
| TOTAL | **204,080** | 6,652 | 9,337 | 12,656 | **28,645** |

**Table 1: Footprint (arm - in bytes) at peak usage**

### 8.1 Footprint

The system can be compiled with many compilers. The measurements were carried out using the ARM Software Development Kit 2.11. Table 1 shows the memory usage of the whole system. The ROM footprint is the amount of Flash required. The RAM footprint is measured at the point of execution where the memory usage was at its maximum. The RAM usage of the individual components varies but this is the point that determines how much actual RAM is required.

Figure 7 and Figure 8 show the percentage occupation of footprint. The network stack occupies about 38% of the footprint of the whole system. It includes DHCP, IGMP, IP, UDP, multicast, routing, sockets, etc. The code also supports IP auto-configuration [21] for IP applications on an ad hoc network in the absence of a DHCP server.

The XML parser and generator take about 8KB or 4% of the total. The SOAP component includes a schema checker and a serializer/deserializer that translates between SOAP messages to application stack frames according to an XML specification.

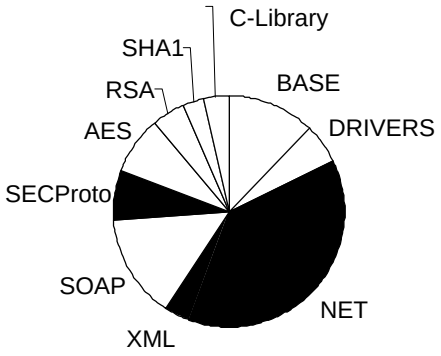



**Figure 7: ROM Footprint**

The SECProto component leverages the SOAP component and includes the security protocol implementation, i.e. trust management, service discovery, key distribution, two-way authentication, etc. The cryptographic algorithms AES, RSA and SHA-1 are lifted from Windows® and are not particularly optimized for size. The C-Library is the part of the ISO C runtime library that was used.

The BASE component includes the real-time scheduler, a heap manager, a loader, any machine dependent initialization code, threading and synchronization, and the unified namespace.

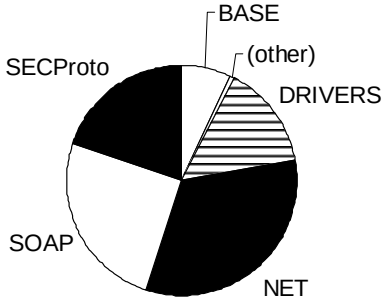



**Figure 8: RAM footprint at peak usage**

The total system heap is 9,337 bytes including the usage of the heap itself but excluding stacks. We can see that the ROM footprint (including code and read only data) of the whole system is less than 200 KB (204,080 bytes) and at peak usage the RAM footprint—including static data (.data, .bss, and interrupt vectors), heap and stacks—is about 28 KB (28,645 bytes).

### 8.2 Latency

We measured how long it took to respond to a service message. Table 2 shows the latency of major cryptography operations. We ran each operation on an EB63 board—running from slow external Flash memory—500 times and calculated the average latency and the standard deviation. Generating a key pair of 1024-bit RSA takes almost 5 minutes.

Fortunately, key pair generation is required only once on each device, when first initialized.

Encryption/decryption with an RSA private key is also quite expensive. It takes about 103 seconds per kilobyte. There are two applications of RSA: 1) signing the hash value of a certificate and 2) exchanging a peer-to-peer key. For both of these cases, the data is less than a block (128 bytes). A certificate only needs to be signed once. Any later use involves the cheaper public key operation. The latency of signing a certificate is thus about 14 seconds.

Exchange of one peer-to-peer key needs four RSA cryptographic operations: Two with private keys and two with public keys. The first private key operation can be pre-calculated so that it does not need to be factored into the response time. Therefore, the cryptographic latency of peer-to-peer key exchange is about 14 seconds. If one of the parties is a more powerful computer, it can bear the burden of both expensive operations by a combination of RSA and DSA use. That cuts the exchange time down to about 1.5 seconds.

The most frequently used cryptographic operations are AES encryption/decryption and SHA1-HMAC. Table 2 shows that encrypting/decrypting one KB with 128-bit AES takes about 16.3 ms and hashing one KB with SHA1-HMAC takes about 79.6 ms. However, in AES CTR mode, counters can be predicted easily so that encrypting the counters with an AES key can be pre-calculated during CPU idle time. Thus the latency of CTR-mode AES cipher is just XOR operation time. XOR operation latency is negligible. Therefore, the latency to encrypt a message with one KB and hash it is just 79.6 ms.

Table 3 shows that the latency of doing one remote ADD operation through SOAP. The SOAP request message for ADD operation is 835 bytes long and the SOAP response message has 747 bytes.

Including ~74 bytes of overhead (14-byte Ethernet header, 20-byte IP header, 8-byte UDP header, and 20 bytes for HMAC, 4 for Sequence number and 16-byte alignment) for each of them, the serial packet for the SOAP request message is 912 bytes, and the SOAP response message is 818 bytes. Transmitting one byte on serial line needs 10 bits. Therefore, the theoretical serial transmit latency = (912 + 818) * 10 / 38400 / 1000 = 450 ms, where 38400 is the baud rate of the serial line.

### 8.3 Energy Consumption

Energy consumption is directly related to: a) how much data has to be transmitted—the time the radio is on, and b) the amount of computation that needs to be done—the time the CPU is on. Some of the cost could be alleviated by compressing the messages so less is transmitted.

The overhead of the secure protocol is, on average, 30 bytes per service message compared to plain text messages. This is about 4% of the XML messages. With a compressed message, the overhead would be somewhat higher.

The CPU overhead of the encryption per message with the same dataset as the latency calculation is about 20%, excluding the one time costs of certificate handling and key exchange. This is the percentage of the message processing—the percentage of the total workload of the system depends

| Algorithm | Operation | | Latency on a 25 MHz ARM 7 | | |
|---|---|---|---|---|---|
| | | | Average | Standard deviation | Per KB |
| 1024-bit RSA | Generate a key pair | | 290 s | 56% | N/A |
| | Private key | Encrypt/decrypt a block (128 bytes) | 12.9 s | <1% | 103 s |
| | Public key | Encrypt/decrypt a block (128 bytes) | 0.667 s | <1% | 5.34 s |
| 128-bit AES | Encrypt/decrypt a block (16 bytes) | | 0.254 ms | <1% | 16.3 ms |
| SHA1-HMAC | 1024 bytes | | 79.6 ms | <1% | 79.6 ms |

**Table 2: Latency of cryptography operations**

| | Latency on a 25 MHz ARM 7 | | Processes included |
|---|---|---|---|
| | Average | Standard deviation | |
| Total measured latency | 760 ms | 5% | Generate, parse, process, encrypt/decrypt and transmit the SOAP request and response. |
| Theoretical serial transmit latency | 450 ms | N/A | Ideally transmit the request and response on serial line. |
| Local SOAP processing latency w/o encryption | 101 ms | 2.1% | Generate, parse and process the SOAP request and response. |
| Cryptographic latency | 65.6 ms | <1% | AES and SHA1-HMAC on the request and response. |
| Other | 143 ms | | Drivers, network stack, etc. |

**Table 3: Latency of remote ADD operation**

PRELIMINARY

on applications that run on the platform. We measured the EB63 board energy usage and observed that it consumes 68 mA at 7V when idle and 108 mA when busy. Integrating the difference for one service request yields 270 mJ, which corresponds to 20 million cycles. More energy efficient hardware would yield smaller numbers. The measurement, however, reflects the finding that the protocol processing can be done within reasonable time and within a reasonable number of cycles.

## 9 RELATED WORK

Many researchers identify that trust management plays a significant role in a distributed security system [22][23] [24]. Yahalom *et al.* gave a formal definition of trust [40]. Wilhelm *et al.* discussed trust management for mobile networks in [18]. A trust bootstrapping protocol, the Resurrecting Duckling protocol, is proposed in [5]. It avoids an online CA and is suitable for low-cost device use. We extend the Resurrecting Duckling protocol to function relationship initialization and trust federation.

Tatebayashi *et al.* proposed a key distribution protocol (TMN) for mobile network [31]. However, their protocol is only suitable for star-type mobile networks and some researchers point out it is flawed, e.g. Simmons describes an attack against TMN [39] and Park *et al.* also analyzed its weakness and propose an improvement for it [33]. Carman *et al.* compared performance of a wide variety of key distribution schemes on different sensor network platforms [8].

Beller and Yacobi propose a protocol that uses pre-computation techniques to reduce the response time of key distribution for mobile uses [37]. However, their protocol is vulnerable to a man-in-the-middle attack [36].

Zhou and Haas use routing redundancies of ad hoc networks to achieve availability, and use threshold cryptography to isolate compromised nodes [32]. Marti *et al.* proposed a mechanism that uses a watchdog to recognize misbehaving nodes and then uses a *patherater* to avoid them.

Hubaux, Buttyan and Capkun analyze security threats specific to ad hoc networks and propose a self-organizing public-key distribution scheme, in which certificates are issued by users (corresponding to devices in our work) instead of a CA [35]. Their algorithm involves complex graphic operations that are neither scalable nor suitable for embedded systems use.

Czerwinski *et al.* propose a secure centralized service location model, in which service advertisements and queries are all done through a central server [9]. However, their work is too complicated for use on low-cost devices.

All the above papers involve one or several aspects of the security issue for low-cost embedded system use. The model proposed in this paper pulls them together and fills in the gaps.

Fox and Gribble presented a security protocol for mobile computing based on a proxied version of Kerberos IV, which provides secure access to application level services [11]. However, as would be expected, their solution requires an online centralized authentication service. Traditional security solutions that require online trusted authorities or certificate authorities are not suited for mobile ad hoc network environments. Mobile ad hoc networks are often unable to provide access to an online centralized trust authority due to their highly dynamic infrastructure and their need for reliable autonomous operation.

Perrig *et al.* [6] propose two secure building blocks for low-cost devices: SNEP and $\mu$TESLA. They claimed that SNEP provides confidentiality, authentication, and data freshness, and $\mu$TESLA provides authenticated streaming broadcast. Their system security is based on a preset master key shared by all devices. The block chaining they use, does not remove the weakness that if an adversary compromises the master key on any device, he can easily eavesdrop and impersonate all other devices. In our system, we use PKI to exchange trust and peer-to-peer keys and avoid these pitfalls, but end up with a slightly more complex model. Their savings from avoiding the trust and key issues do not appear to help—there still is no space left for any applications in [6].

We introduced embedded Web Services in [3]. No other comparable work has been published since. In this paper, we focus on secure Web Services for invisible computing use. As far as we know, even though recently much work has been done on ad hoc network security, nobody else has done any research on secure Web Services for embedded systems. Our work shows two suppositions are incorrect: 1) Web Services would be unsuitable for embedded use because they need a large footprint, CPU time, energy, and network bandwidth; and 2) that public key infrastructure (PKI) would be unsuitable for embedded use because it consumes lots of energy and CPU time [6].

One may argue that Web Services' advantages come with a performance penalty: XML based SOAP messages are textual so that their sizes are significantly larger than protocols that send specific binary data. It turns out that the special protocols are often not that efficient and their inability to scale to new demands make it necessary to support many different mechanisms largely erasing any imagined performance benefit. We also note that compressing XML can be done in CPU-efficient ways and result in significant reduction in size.

## 10 CONCLUSIONS

This paper described a secure communication model and implementation for invisible computing. It showed that it is possible to combine low-cost with strong security and first class interoperation. A trust and key exchange model based on public key infrastructure and a presentation layer based on XML Web Services were not out of reach when properly put together.

Combining trust establishment with functional assignment led to a physical touch based user interaction paradigm that did not completely eliminate configuration, but made it simple and understandable. Federating with outside trust authorities proved centralized and hierarchical models unnecessary. The independence achieved allows for incremental and self-sufficient deployment.

The security does not come for free but the cost is in our view reasonable considering the alternative of inadequate security. A high level of security and interoperation is achievable on low-cost devices and should therefore be adopted.

The implementation proved that secure Web Services make sense in embedded systems with a careful design and on-the-target optimizations. The disciplined component-based approach of MMLite made it a good platform for achieving the efficiency goals and enabled the project to be completed.

## ACKNOWLEDGEMENTS

Thanks to Tuomas Aura and Yacov Yacobi for advice on the cryptography and security protocols.

## REFERENCES

[1] United States House of Representatives, Committee on Energy and Commerce, Subcommittee on Oversight and Investigations, "Statement of Dr. Samuel G. Varnado – Sandia National Laboratories,"

http://www.sandia.gov/news-center/esources/congress-testimony/ pdf/Varnado020709.pdf

[2] Johannes Helander, Alessandro Forin, "MMLite: A Highly Componentized System Architecture," in *the 8th ACM SIGOPS European Workshop*, September 1998.

[3] Alessandro Forin, Johannes Helander, Paul Pham, Jagadeeswaran Rajendiran, "Component Based Invisible Computing," in *the 3rd IEEE/IEE Real-Time Embedded Systems Workshop*, London, December 2001.

[4] A. Abdul-Rahman, S. Hailes, "A distributed trust model," in *Proc. New Security Paradigms Workshop* (*NSPW-97*), New York: ACM, 1997, pp. 48-60.

[5] F. Stajano, R. Anderson, "The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks," *LNCS 1796*, Springer-Verlag, 1999.

[6] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, J. D. Tygar, "SPINS: Security Protocols for Sensor Networks," in *Wireless Networks Journal* (WINE), September 2002.

[7] "Security in a Web Services World: A Proposed Architecture and Roadmap,"

http://www.verisign.com/wss/ architectureRoadmap.pdf

[8] David W. Carman, Peter S. Kruus, Brian J. Matt, "Constraints and Approaches for Distributed Sensor Network Security," NAI Labs Technical Report #00-010.

[9] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, R. H. Katz, "An Architecture for a Secure Service Discovery Service," in *the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking* (*MOBICOM 1999*), pages 24-35, Seattle, WA USA, August 1999.

[10] IBM Corporation and Microsoft Corporation, "Federation of Identities in a Web Services World,"

http://msdn.microsoft.com/webservices/understanding/advancedwebservices/default.aspx?pull=/library/en-us/dnglobspec/html/ws-federation-strategy.asp

[11] Armando Fox, S.D. Gribble, "Security on the move: indirect authentication using Kerberos," in *the 2nd Annual ACM/IEEE International Conference on Mobile Computing and Networking* (*MOBICOM 1996*), pages 155-164, White Plains, NY USA, November 1996.

[12] Bahrat Patel, Jon Crowcroft, "Ticket based service access for the mobile user," in *the 3rd annual ACM/IEEE International Conference on Mobile Computing and Networking* (*MOBICOM 1997*), pages 223-233, Budapest Hungary, September, 1997.

[13] "PKCS#1 v2.1: RSA Cryptography Standard," RSA Laboratories, June 14, 2002.

[14] P. Cheng, R. Glenn, "HMAC: Keyed-Hashing for Message Authentication," *Internet RFC 2104*, February 1997.

[15] Whitfield Diffie, Martin Hellman, "Privacy and Authentication: An Introduction to Cryptography," in *Proceedings of the IEEE*, 67 (1979), pp. 397-427.

[16] Morris Dworkin, "Recommendation for Block Cipher Modes of Operation – Methods and Techniques," *NIST Special Publication 800-38A, 2001 Edition*.

[17] Srdjan Capkun, Levente Buttyan, Jean-Pierre Hubaux, "Self-Organized Public-Key Management for Mobile Ad Hoc Networks," *IEEE Transactions on Mobile Computing*, vol. 2, no. 1, January-March 2003.

[18] U. G. Wilhelm, S. Staamann, L. Buttyan, "On the problem of trust in mobile agent systems," in *IEEE Network and Distributed Systems Security Symposium 1998*, pages 11-13, San Diego, CA.

[19] T. Grandison, M. Sloman, "A Survey of Trust in Internet Applications," in *IEEE Communications Surveys*, Fourth Quarter 2000.

[20] S. M. Bellovin, M. Merritt, "Limitations of the Kerberos Authentication System," in *ACM Computer Communication Review* vol. 20(5), pp. 119-132, 1990.

[21] Ryan Troll, "Automatically Choosing an IP Address in an Ad-Hoc IPv4 Network,"

[22] DoD Trusted Computer System Evaluation Criteria, 26 December 1985 (Supercedes CSC-STD-001-83, dtd 15 Aug 83). (Orange Book)

[23] Matt Blaze, Joan Feigenbaum, Jack Lacy, "Decentralized Trust Management," in *Proceedings of the IEEE Conference on Privacy and Security*, 1996.

[24] P. Zimmermann, "PGP User's Guide," *MIT Press*, Cambridge, 1994.

[25] IBM®, Microsoft®, etc., "Specification: Web Services Federation Language (WS-Federation), 08 July 2003,"
http://www-106.ibm.com/developerworks/webservices /library/ws-fed/

[26] "Web Services Architecture—W3C® Working Draft 8 August 2003," http://www.w3.org/TR/ws-arch/

[27] "Simple Object Access Protocol (SOAP) 1.1—W3C® Note 08 May 2000," http://www.w3.org/TR/SOAP/

[28] "SOAP Version 1.2 Part 1: Messaging Framework—W3C® Recommendation 24 June 2003,"
http://www.w3.org/TR/soap12-part1/

[29] "SOAP Version 1.2 Part 2: Messaging Framework—W3C® Recommendation 24 June 2003,"
http://www.w3.org/TR/soap12-part2/

[30] "Extensible Markup Language (XML),"
http://www.w3.org/XML/

[31] M. Tatebayashi, N. Matsuzaki, D.B.J. Newman, "Key distribution protocol for digital mobile communication systems," in *Advances in Cryptology-Crypto '89 Proceedings*, Lecture Notes in Computer Science, vol. 435, 1989, pp. 324-334.

[32] L. Zhou, Z. Haas, "Securing ad hoc networks," *IEEE Network Magazine*, 13(6), 1999.

[33] C. Park, K. Kurosawa, T. Okamoto, S. Tsujii, "On key distribution and authentication in mobile radio networks," in *Advances in Cryptology EuroCrypt '93*, Lecture Notes in Computer Science, vol. 765, pp. 461-465, 1993.

[34] S. Marti, T. Giuli, K. Lai, M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks," in *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking*, 2000, pp. 255--265.

[35] J. Hubaux, L. Buttyan, and S. Capkun, "The quest for security in mobile ad hoc networks," in *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing* (MobiHOC) 2001.

[36] C. Boyd and A. Mathuria, "Key establishment protocols for secure mobile communications: A selective survey," in Proceedings of ACISP'98 , Lecture Notes in Computer Science, vol. 1438, 1998, pp. 344-355.

[37] M. J. Beller, Y. Yacobi, "Fully-Fledged Two-Way Public Key Authentication and Key Agreement for Low-Cost Terminals," *Proceedings of Electronic Letters*, May 27, 1993, Vol. 29, No. 11, pp. 999-1000.

[38] S. Basagni, K. Herrin, E. Rosti, D. Bruschi, "Secure Pebblenets," in *ACM International Symposium on Mobile Ad Hoc Networking and Computing* (MobiHoc), pages 156--163, 2001.

[39] G. J. Simmons, "Cryptanalysis and protocol failure," *Communications of the ACM*, 37(11), Nov 1994.

[40] R. Yahalom, B. Klein, T. Beth, "Trust relationships in secure systems—A distributed authentication perspective," in *Proc. of the 1993 IEEE Symposium on Research in Security and Privacy*, pages 150-164, May 1993.

[41] A. J. Menezes, P. C. Oorschot, S. A. Vanstone, "Handbook of Applied Cryptography," CRC Press, 1997.

[42] "AT91EB63 Evaluation Board User Guide,"
http://www.atmel.com/dyn/resources/prod_documents/DOC1359.PDF

[43] "AT91M63200 Summary, AT91 ARM Thumb MCU,"
http://www.atmel.com/dyn/resources/prod_documents/1028S.PDF